JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 1321-1336 (2009)

Service-Centric Integration Architecture for Enterprise Software Systems

CLAUS PAHL*, WILHELM HASSELBRING[1] AND MARKUS VOSS[2]
*School of Computing
Dublin City University
Dublin 9, Ireland
E-mail: Claus.Pahl@dcu.ie
[1]Software Engineering Group
University of Kiel
D-24118 Kiel, Germany
E-mail: wha@informatik.uni-kiel.de
[2]Capgemini sd&m AG
63065 Offenbach, Germany
E-mail: markus.voss@capgemini-sdm.com

Service-oriented architecture (SOA) has the potential to provide solutions for enterprise application integration (EAI) problems. While core service platform technologies exist, methodological approaches that link the business domain with the platform infrastructure – a prerequisite for an EAI solution – are only beginning to mature. We present a framework for integrating service-centric software systems that emphasizes service architecture as the key to the application integration problem and that proposes a service-centric integration architecture layer to coherently bridge the gap between business and platform layers. Domain models act as drivers of the architectural development and integration process. A dedicated and empirically developed service-specific architecture solution, based on an adaptation of the QUASAR Enterprise methodology, utilizes service and architecture identification techniques.

Keywords: service-oriented architecture, enterprise application integration, architecture description, integration architecture, service process modelling

1. INTRODUCTION

An enterprise architecture describes an enterprise-wide system of information systems. The methodological context of enterprise application integration (EAI) with its domain engineering and application engineering techniques dominates development and integration solutions [1]. The terminology already points to architecture as the key to a solution for the integration problem. The difficulty, however, lies in the heterogeneity of the problem space in several dimensions. Application platforms and languages as well as modelling notations and methodologies often vary.

With the emergence of Web Service technology as a platform and service-oriented architecture (SOA) as the corresponding methodological context, the integration problem has found a solution [2]. Service-centric software applications and systems can easily be assembled, integrated or migrated. While platform technology is maturing and has al-

Received October 31, 2007; accepted June 27, 2008.
Communicated by Jonathan Lee, Wei-Tek Tsai and Yau-Hwang Kuo.
* Corresponding author.

ready demonstrated its potential, an architectural service engineering approach that links the enterprise-level business context with platform considerations is still lacking. An approach beyond current best practice is needed to meet the need for automated development support to achieve improved consistency and cost-effectiveness qualities.

Integration architecture is at the core of our solution, emphasizing service and process-centric architectures as the key to our solution. We present a service-centric EAI framework – a layered modelling and architecture approach consisting of empirically developed techniques and methods, which are adapted to the services context. Specifically, we discuss suitable notations for layered integration architecture description and the transformation between these layers. business model-driven service architecture provides the necessary coherence across the different development stages. We have based our framework on an adaptation of the successfully employed QUASAR Enterprise methodology for application integration [3].

The proposed focus on architectural modelling with an integration architecture layer at the centre results in a number of benefits:

- It provides a coherent solution that integrates modelling concerns ranging from business processes and business domain models to application infrastructures to service-level software components [3] by mapping these concerns into a process-centric architecture modelling notation. Coherence enables higher degrees of integrity among models and consequently delivers improved reliability of resulting software applications.
- The utilization of the recently standardized Business Process Modelling Notation BPMN [4] provides us with a notation that, firstly, has the potential to be used beyond IT specialists by the business community and, secondly, is supported by predefined transformations. It serves us as an interoperable notation for the central software architecture concerns and also allows us, due to its consistent use throughout, to automate central activities.

Overall quality improvement in terms of system maintainability, achieved through enhanced architecture models, and the cost-reduction through tool-supported languages and the automation of important activities are central goals.

Our contribution comprises model-based notations and techniques for service-centric integration architecture for enterprise application integration. A service- and process-based focus throughout enables the integration of business domain modelling and application architecture modelling. Core elements of our contribution are:

- A layered architectural modelling language based on a BPMN extension is the key to improved quality.
- Two central architectural modelling activities characterize the approach: service identification based on business process activities and domain-specific concerns and integration architecture identification based on a reference model organizing services into layers and categories.

A central role supporting the two identification steps is played by the information architecture. The information architecture provides domain-specific solutions to service identification at both the business and platform level and architecture identification.

2. ENTERPRISE APPLICATION INTEGRATION AND SERVICE ARCHITECTURE

Enterprise Application Integration (EAI) [1] is a central concern of large-scale software systems that has gained importance rapidly due to the recent emergence of service-oriented architecture (SOA) as an architecture and development framework for service-centric software systems. SOA provides service orientation as a structuring principle that can form the backbone of EAI solutions [5]. Enterprise IT architectures consist of complex systems of interdependent components that are governed by business processes [6]. The logical integration of information and process aspects is the central aim of EAI [7]. Software architecture is the ideal location to place an integration solution [8]. Service-oriented architecture links integration to a development process, but also to a platform that aims at interoperability in heterogeneous environments.

Consistency of information and uniformity of access are usually the goals of EAI, but also evolution, maintenance and interoperability are drivers of EAI. Different layers of integration, from information to services to presentation, implement EAI. In the SOA context, service-based integration becomes the central driver of integration needs. With SOA as the EAI approach, an integration architecture technique becomes the key to the solution of the problem. Integration architecture defines the stage and the location at which service integration is realized. Integration architecture is framed by two activities in the service engineering process model:

- Business architecture modelling aims at capturing principles of the application domain. Business processes that provide business services are part of this aim, as are more product-oriented and organizational views on the business domain.
- Service implementation development addresses the implementation and representation of services within a given platform, such as Web Services technologies.

Business architecture modelling and service implementation development are, however, not the focus of this paper. In order to provide an integration architecture solution, we need bridge the gap between business models and service implementation through a SOA-specific architecture solution. We propose to incrementally transform business-level models towards service architecture descriptions.

Integration architecture is a SOA-based EAI approach based on a number of service-centric architecture and modelling techniques, see Fig. 1. We propose layered activities corresponding to the layers of business process integration, application integration and platform-oriented systems integration as established in the area of information system integration – see [7] for details. The proposed incremental approach is based on two techniques:

- the identification of services based on activities of a business process model, which is a central concern of application integration in order to consistently integrate and manage software applications,
- the identification of an architecture that integrates services of an application in a coherent platform-oriented system architecture that allows the implementation using Web service technologies.

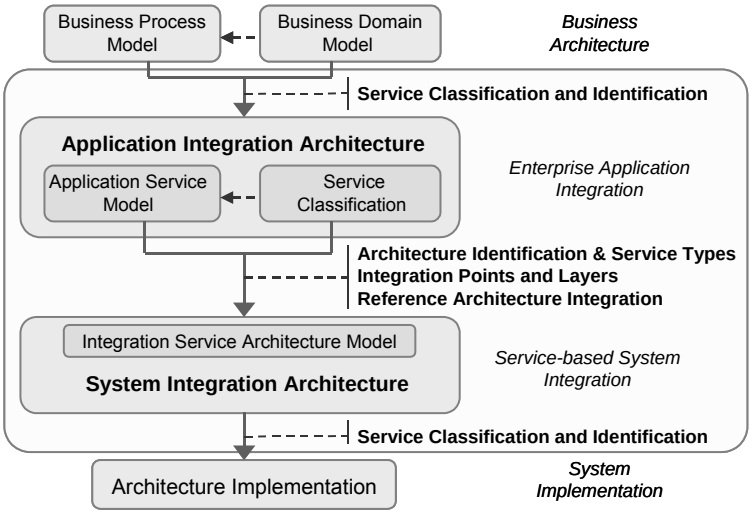

Fig. 1. The two central integration architecture layers (application and system) and integration architecture activities for the two layer transformation stages.

Based on architecture constraints and service analysis, classification and identification techniques, we incrementally add architectural detail to the models such that an integration of systems on the service platform can be implemented. The service identification based on the application domain layer and service-based architecture integration in the integration layer are addressed in sections 3 and 4.

3. SERVICE IDENTIFICATION

A comprehensive account of an application context with business architecture views such as business processes and the business domain, but also a structural and behavioural view of the application architecture is an essential prerequisite for integration architecture. A business architecture model provides a process-independent information architecture that acts as a constraining vocabulary based on domain concepts and activities for an application domain.

This section introduces our notational framework, an extension of the Business Process Modelling Notation BPMN [4], and a technique for service identification based on this notation to support application integration.

3.1 Business Architecture Modelling with Extended BPMN

Service-oriented architecture is about services, and, essentially, about the composition of services to processes. Orchestration and choreography are two perspectives on service process composition. Processes are also at the centre of the business domain. Business processes provide structure and organize an enterprise. These processes are embedded into a business domain context defining the organizational constraints and the product-oriented perspective.

BPMN is a notation to model business processes [4]. Crucial to our approach is the utilization of BPMN as an architecture description language, *i.e.* as a notation that defines the components (here services at various levels of abstraction), their connections, and the static and dynamic dependencies between them. BPMN's process-centricity makes it an ideal candidate for service architecture issues such as orchestration and choreography as two ways of expressing service connectivity and dependencies [2]. The standardization of BPMN and support in terms of defined transformations into WS-BPEL [9] make it suitable to achieve the desired degree of automation for a fully service-centric development approach. We have chosen BPMN over executable notations as executability is not required and a notation is needed that is acceptable to software architects as well as business analysts.

We extend BPMN and provide a meta-model layer that captures product and organizations aspects. Two elements are added:

- a separate domain ontology in the form of a concept taxonomy that defines and organizes the objects (such as products) and activities (such as processes) of the domain that are used in business process models,
- an integration of process models and their activities into the concept taxonomy, providing a semantic context for each process.

The business domain model is illustrated by an enterprise organization taxonomy in Fig. 2, which shows an excerpt of a sample hierarchy for a policy application processes for the insurance domain. A taxonomy is a hierarchical representation of concepts of a domain, ordered using the subsumption relationship between concepts. We use the insurance domain through to provide examples for notations and techniques. The insurance domain is an application context in which our underlying Quasar architecture methodology has been extensively applied [3].

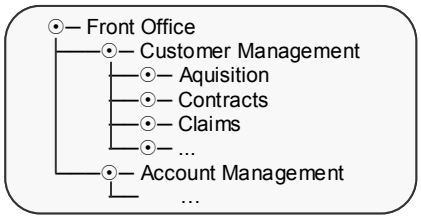

Fig. 2. Ontology-based domain taxonomy model for an insurance application.

This domain model extension of BPMN provides essentially a simple ontology – a vocabulary with a hierarchical structuring mechanism – that captures organizational units, their products and activities. Business processes at different levels of abstraction are embedded into this hierarchy. We use BPMN as a conceptual process modelling language that allows the embedding of processes into their organizational context – see upper left corner in Fig. 3, which is a excerpt of the domain ontology in Fig. 2.

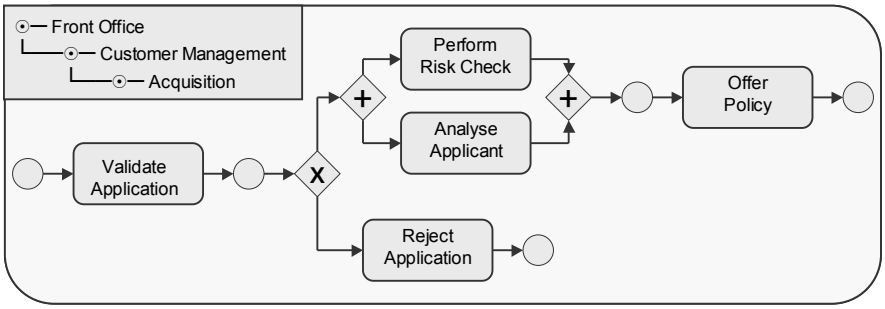

Fig. 3. Insurance policy application business process in an insurance context.

3.2 Service Identification at the Application Integration Layer

The business architecture models provide a computation-independent architectural perspective. Crucial is now to bridge the gap between this and a system-oriented view. We propose to first create an application integration layer, which acts as a development plan for the overall system architecture, *i.e*. firstly, it captures application integration through services that have been identified based on business-level constraints and, secondly, is going to prepare the system-oriented integration architecture stage. The functionality of the system shall be elicited in terms of the processes and organizational constraints identified in the business domain model. The development plan prepares for detailed architectural decisions at lower layers.

The architectural concern at this stage is what is often called IT architecture. It aims at a holistic view of a service-centric software system in the context of its governing business aspects and the overall software systems infrastructure. The central difficulty is how to utilize the business-level models to create an architecture description. We propose a method for this stage adding service identification and classification to the business-level models. Three individual application integration tasks can be identified:

1. service classification based on domain-driven classification criteria,
2. service identification of application-level (usually coarse-grained) services,
3. service hierarchy definition to refine coarse-grained services.

The classification is based on business facets – specifically the product and process dimensions. The instrument to carry out the service identification and their classification is a classification matrix, presented in Fig. 4, where it is applied to the insurance domain to identify and organize services. The two dimensions result from the product (business domain) and process (business process) aspects of the business architecture models. These two dimensions themselves are generic and independent of the domain itself. The elements in each of the classification dimensions are domain-specific and are based on the domain information taxonomy. We have defined similar classification and identification matrixes for other domains such as banking [3] – based on the experience and expertise of our industrial partner in a large number of integration projects. The services identified through the classification are generally broad and often refined in further system-oriented steps. The aim of this classification is application integration, not software system-level service definition.

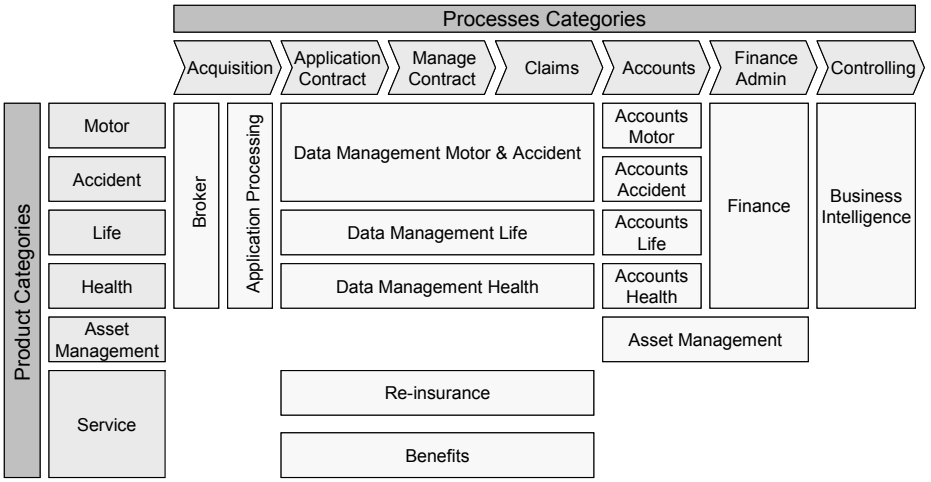

Fig. 4. Service classification and identification for an insurance application.

The identified services, such as the account or data management services, are classified according to the two generic dimensions, see Fig. 4 where domain concepts such as products and processes from the business models define the dimensions. The domain taxonomy model can even provide the information to structure identified services hierarchically. Reference architectures play an important role here to constrain the classification and organization. Detailed organization models often stem from reference architectures for vertical domains, such as banking or insurance. For instance, a life insurance policy creation service could be a subservice of the data management process for life insurance, which in turn is a subservice of a policy management process.

The step from the business layer to the application integration architecture is a refinement step. The process model, see *e.g*. Fig. 3, is still the central architectural description. This BPMN-based process model is, however, complemented by a service classification meta-model (Fig. 4) that, as just illustrated, categorizes services and that includes individual services and processes into a hierarchy, which is often dominated by reference architectures. The application processing activity described in Fig. 3 is categorized using the Fig. 4 scheme as belonging to the acquisition process category and as spanning several product categories. In terms of concerns of architectural description languages, this meta-model resembles an architectural type language that adds a semantical layer to process definitions and serves to constrain an extended BPMN model for further architectural concerns.

4. SERVICE-CENTRIC ARCHITECTURE IDENTIFICATION

Integration architecture addresses the difficult issue of defining a basis on which heterogeneous application components can be integrated. It needs to provide the abstract frame in which existing and new systems can be integrated, in which legacy systems can be accessed in a uniform way and into which legacy systems can be migrated. Based on our experience, coarse-grained services that can capture a number of applications (legacy

or to be developed/acquired) at the systems level are a suitable starting point. We have distinguished an application and a systems layer for integration architecture. The business architecture models constrain the integration from the business perspective. Two central decisions, which would have to be considered in a generic approach to EAI, have already been taken:

- Services as the organizing principle of software is currently considered the best solution to the integration problem [10].
- Web Services are currently the predominant platform for interoperable service deployment [11].

4.1 Service-centric System Integration Architecture

The architectural development plan that the application integration architecture describes need to be applied to define a coarse-grained system architecture design that addresses the following issues:

- system boundaries that separate the system focus from its supporting environment,
- interfaces that clearly identify structural and behavioural dependencies between services,
- higher-level assemblies of services through a component mechanism.

The application integration architecture defines a development plan that guides the development of a coarse-grained architecture, whose purpose it is to provide an abstract, logical integration layer and its link to a supporting platform. We propose an integration architecture development method in three steps:

1. architecture identification and service types;
2. service-based integration;
3. logical architecture identification.

The first step refines the initial service classification from the application integration discussed in section 3 and adds necessary software architecture and system-related information. Essentially, the business-centric notions of process and activity have to be reinterpreted as software system-specific concepts, *i.e.* this is a change of focus from computation-independent to platform-independent architecture modelling and further on to platform-specific implementation.

We can distinguish two layers of the service-centric integration architecture model – a logical, platform-independent layer and a physical, platform-specific layer.

- The purpose of the logical layer is the platform-independent orchestration of services as elements of composite processes. Its conceptual elements are flows that model processes, whereby the service notion is viewed as conceptual. We can still use the same extended BPMN notation, but now in a more structured way and interpreted in terms of logical system architecture considerations, *i.e.* platform-independent, but not computation-independent anymore. It should, however, be noted that this is not the typical bus-oriented model; essential elements are process and service abstractions.

- The purpose of the physical integration architecture layer is the platform-specific implementation of orchestration. Its conceptual elements are services and processes, where the service notion is directly linked to the Web services platform. We can use predefined translations from BPMN to WS-BPEL in the transformation from the logical to the physical layer. In order to produce executable code, our extensions to BPMN are not needed, *i.e*. we can directly use the predefined transformations. The purpose of our BPMN extension is to support the architecture development and integration through semantic constraints.

The first three steps above relate to the development of a logical system integration layer. A further fourth step is needed to address platform integration in the form of a logical to physical layer transformation, which shall also be discussed briefly to emphasize the automation potential at this stage.

4.2 Architecture Identification and Service Types

The business domain drives this initial architecture identification step. The business models provide context and also the subject of the identification technique. We use a 4-layered categorization and reference architecture scheme for services, see Fig. 5, that assigns service types to services and organizes these into data, function, process and interaction layers. Data services provide access to stored business data. Functional services support specific business functions such as credit checks or customer identification. Process services support a composite business process, usually across several functions. Interaction services coordinate external access to the system services.

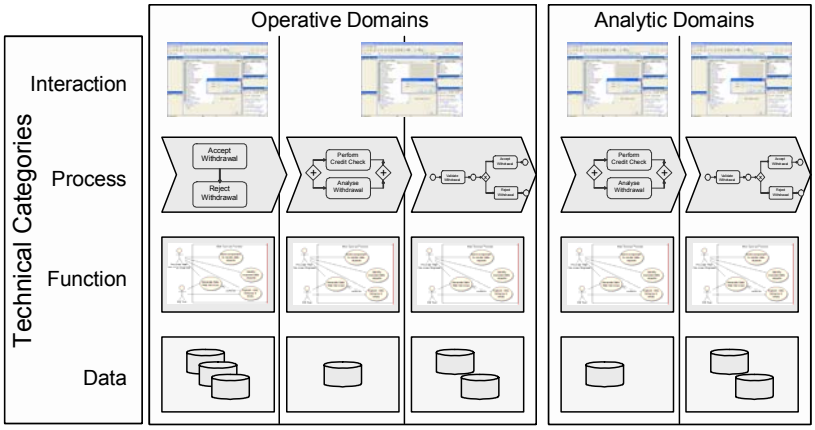

Fig. 5. A 4-layered service classification and reference architecture schema.

The business process elements – including the business activities and processes themselves, but also functionality and data aspects derived from the business domain model – can be classified along these technical categories in order to characterize their role for integration on the system architecture layer through a specific type. The process layer is the starting point of a top-down style architecture identification; it can be derived

directly from the business process model in the first step, then refined and detailed into individual functions. Although the business process models provide input here, a significant amount of architecture-related information (data, distribution, choreography) needs to be added – this is going to be detailed in the next section. The data layer is also a reflection of information models at the business level. The presentation layer is not part of the business-driven integration architecture and needs to be added on top of the service layer.

4.3 Service-based Integration Points and Layers

An integration point of an application enables another application to communicate in terms of data or control flow elements. Data, service, and presentation are three layers on an application that can act as integration points. An integration layer is determined by the primary focus of integration points in terms of the architectural layers of the reference architecture. In the SOA context, services are the primary layer of integration for processes and functionality. Service-based integration through publication of service interfaces and interface-based invocation is the core principle of service centricity.

Detailed service-based control flow needs to be addressed for integration architecture models. Abstractly defined business processes need to be refined or amended to provide the necessary level of detail for the integration architecture focus:

- adding data to indicate the data processing capabilities of individual services,
- separating processes across a possibly distributed deployment topology to reflect logically and physically separate organizational units,
- adding service choreography aspects (*i.e.* message passing between independent processes) to the orchestration focus on business process models.

Fig. 6 presents a refinement of the previous policy application process at the process layer of the reference architecture schema, adding data (a policy application), distribution (two swim lanes), and choreography (interaction across swim lanes). We have mapped the withdrawal business process activities onto services, here at the process layer in terms

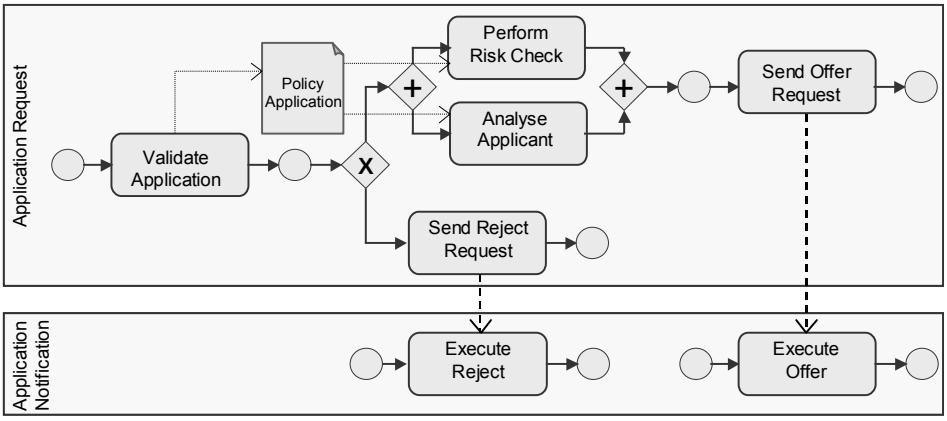

Fig. 6. Logical integration architecture model with data, distribution and choreography aspects.

of our previous categorization. It should be noted that BPMN already allows objects such as documents process separation to be modelled, but it is important here it reinterpret the business-level modelling elements in more computation-oriented terms. The aim is here to obtain a platform-independent representation that can be directly mapped to data representation, message passing and Web service choreography and distribution notations (such as WS-BPEL or WSDL) at the platform level.

Data, distribution and choreography are central software architecture aspects. These aspects help to determine, firstly, function and their data processing abilities for the layer two layers and, secondly, support the transformation into WS-BPEL skeletons based on the additional distribution and choreography information.

4.4 Architecture Definition and Architectural Constraints

The use of reference architectures at this stage is an integral element of integration architecture. In contrast to our earlier use of a layered reference architecture to identify a coarse-grained initial architectural template, a reference architectures at this next step will define common service categories that are typically found in supporting middleware and platform infrastructures. Their deployment as architectural styles (or architectural constraints) is crucial for the architectural modelling of an application in terms of an integrating architecture that is linked to an underlying platform [12]. The organizational perspective of such a reference architecture for an integration platform defines two aspects:

- It defines the structural perspective by providing a subclass hierarchy of nested service categories, see Fig. 7. These serve as component types that can be applied to services.
- The structural perspective is complemented by the service connectivity. Fig. 7 shows an example for the runtime service level where two (unnamed) connections between services are defined.

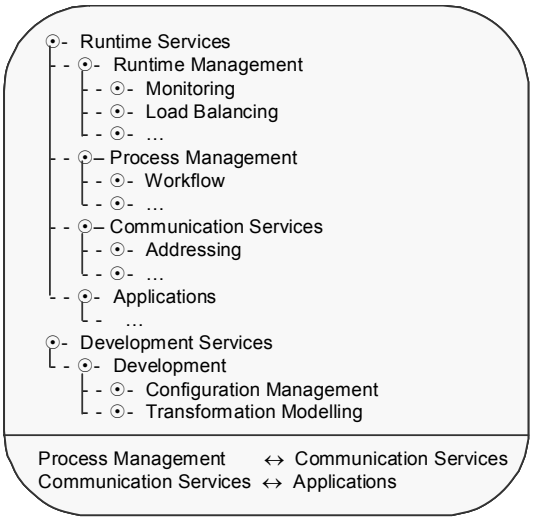

Fig. 7. Reference architecture constraints – Organizational and connectivity perspective on platform-oriented runtime services.

We have again, as for the domain taxonomy, used an ontology-oriented representation for these reference architecture aspects. We have presented these architectural aspects in simplified textual form for the sake of brevity – a representation in terms of BPMN grouping and abstraction mechanisms would also be possible.

The sample policy application process would need to be integrated into the application architecture, which of course would utilize communications or runtime and process management services that are defined in the reference architecture.

Product reference architectures can help match services with tools and service middleware functionality. Most major providers of enterprise software components provide a map of products that can supplement reference architectures.

4.5 Platform Integration

The last step is, at least in the current technological situation, straightforward. WS-BPEL is the predominant orchestration language for Web services as the platform. The BPMN standard defines a translation from BPMN to WS-BPEL, which implements abstract business processes as executable service-based processes in the Web service framework, assuming that concrete, provided services are available for each of their abstract counterparts [13]. Other platforms and middleware technologies such as CORBA may also be used to support the platform implementation.

For the transformation, we do not have to consider the additional architectural constraints such as reference architectures, since these are supplementary constraints that are already satisfied if the required concrete services can be associated to the abstract services. The previous steps guarantee consistency of the logical system integration architecture with models at higher level architecture models.

5. DISCUSSION

The presented framework is the result of an empirically developed methodology. It is an adaptation of the QUASAR Enterprise architecture methodology [3, 14], which has been successfully used in large-scale projects across banking, insurance and automotive domains, to the specific needs of service engineering and Web services as the platform. Quasar Enterprise is itself the result of applying the Quasar architecture approach to quality architectures to service-oriented application integration [14]. Quasar Enterprise is used by sd&m, a software solution and IT consultancy provider with more than 1400 employees, which is active in central Europe.

We evaluated our service-based architecture approach for EAI thoroughly. The Quasar Enterprise approach has been used in sd&m projects of in total more than 1000 person years. The costs for EAI-based service development can be reduced by up to 40%, as an evaluation over a three-year period of the methodology application demonstrates.

In this investigation, we have looked specifically at the language perspective of Quasar Enterprise and language-based formalized transformation steps. The aim here was to provide a notational framework that provides further cost reductions and also more consistency across the development stages through tool support for modelling and partial automation of activities. This model-driven approach has also benefits in terms of

improved maintainability of the integration architecture, which can be enhanced further if tool support is provided throughout.

One of our central observations here is the need to enhance BPMN to an architectural description language [16] to provide a notational framework for service and architecture identification. Reference architectures, that impose constraints on architectures in the form of structural patterns, dominate this concern. This suggests an investigation of the integration of architectural styles into the approach. Alternatives to a BPMN extension would have involved an extension of UML activity diagrams. However, in the services context, transformations from BPMN to WS-BPEL as the execution language are well investigated [13] and suggest using BPMN as the basis of an architectural extension.

6. RELATED WORK

A number of methodologies for application integration have been suggested, such as SOMA, TOGAF or the Zachman framework [5]. We go beyond these, incorporating modelling support to achieve coherence between business and software aspects, and incorporating architectural abstractions starting at business level, thus enhancing the maintainability of the resulting architecture. We have emphasized the notion of architecture modelling, which goes beyond these approaches in terms of its more technical perspective on architecture description [12], which also brings our approach into the context of model-driven development [15]. We can broadly characterize business architecture modelling as a computation-independent modelling concern. System integration architecture is a system-oriented, but platform-independent concern. Application integration links these two concerns to different architectural layers. Abstraction, one of the core principles of model-driven development, is clearly utilized, whereas full automation, the other pillar of model-driven development, is not an objective. We pursue a refinement approach that bridges different architectural layers, *i.e*. aims at the integration of perspectives and concerns through our incremental architecture approach.

Zdun *et al*. [17] present a business-centric architecture framework, determined by patterns, that forms a reference architecture for SOA-based systems development. We, however distinguish two architectural layers for application and system-level integration. In particular if, as in our case, system-independent integration needs to be addressed, then the information architecture providing domain-specific input into the architecture and integration effort becomes central.

A critical step for the design of service architectures is service identification. The granularity of each service impacts on design principles such as loose coupling, reusability, abstraction, and autonomy. Most SOA design methodologies consider service identification a manual task. Some approaches have proposed semi-automatic strategies to identify the set of software components that will provide the services functionality. In [18] business processes elements and information objects are organized in a matrix whose cell values correspond to weights representing the type of manipulation performed in each process step. The matrix is reorganized using an optimization algorithm to determine clusters, which later will represent software components. The optimization criteria aim at minimal communication between components (loose coupling) and maximum compactness of components (high cohesion).

Authors in [19] introduce a framework with reusable architectural decision models as a design method for service realization. Our methodology can be complemented with this framework for structuring architectural decisions. It can, for instance, support the selection of a particular reference model.

The key findings of this discussion are summarized in Table 1.

Table 1. Summary of comparison.

	Coherence	Architectural Abstraction	Architectural Modelling	Decision Modelling
Application Integration Frameworks [5]	No	No	Yes	Yes
Zdun *et al*. [17]	Yes	Yes	No	No
Albani *et al*. [18]	Yes	Yes	No	No
Zimmermann *et al*. [19]	Yes	No	Yes	Yes
Presented Approach	Yes	Yes	Yes	No

7. CONCLUSIONS

Enterprise Application Integration is a problem that has dominated the enterprise-wide view on IT architectures and software applications for a long time. With the emergence of service-oriented architecture, a new paradigm has emerged for the integration of service-centric software applications. A methodological framework for the development of service-centric software systems that emphasizes the integration architecture layer can achieve enterprise-wide integration of new system features, legacy systems, and off-the-shelf products.

A dedicated and empirically developed service-specific solution, based on the QUASAR Enterprise methodology, that utilizes business process modelling and service composition techniques defines our framework. Service architecture is the key to successful integration. An architectural description language consequently needs to be at the centre of such a framework. The demonstration of the suitability of an extension of BPMN to provide the notational basis for this architecture approach was a central objective. The benefits are increased consistency through a formalization of architectural constraints in an architectural description notation and development cost reduction achieved through a tailored, architecture-centric integration approach. We have identified abstraction mechanisms such as reference architectures and constraints as means to provide a mechanism that allows constraining process-centric architectural models in a refinement-style development approach.

Integration architecture needs to satisfy the needs of two perspectives, the computation-independent application-level view in terms of services on the one hand and software the system-oriented service architecture. We have seen that the development of both views is driven by business-level process and information models. A notion of service architecture integrates the views and consequently emerging layers.

Our discussion of model-driven development indicates the future needs arising from our investigation. Automation is becoming central to, firstly, achieve improved cost-effectiveness, but also to achieve reliability of products and application systems through internal coherence of architectural concerns across modelling layers and stages.

REFERENCES

1. S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch, *Enterprise Application Integration*, Elsevier, Spektrum, 2006.
2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services – Concepts, Architectures and Applications*, Springer-Verlag, Berlin, 2004.
3. A. Hess, B. Humm, and M. Voß, "Rules for high-quality service-oriented architectures (in German)," *Informatik Spektrum*, Vol. 29, 2006, pp. 395-411.
4. Object Management Group, "Business process modeling notation (BPMN)," 2007, http://www.bpmn.org/.
5. D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice Hall, 2004.
6. F. Leymann, W. Reisig, S. R. Thatte, and W. M. P. van der Aalst, "The role of business processes in service oriented architectures," *Dagstuhl Seminar Proceedings*, Vol. 06291, 2006, http://drops.dagstuhl.de/opus/volltexte/2006/832/pdf/06291_abstracts_collection.832.pdf.
7. W. Hasselbring, "Information system integration," *Communications of the ACM*, Vol. 43, 2000, pp. 32-36.
8. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., SEI Series in Software Engineering, Addison-Wesley, Boston, US, 2003.
9. WS-BPEL Coalition, "WS-BPEL business process execution language for web services – Specification version 1.1," 2004, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.
10. M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer, "Service-oriented computing: A research roadmap," *Dagstuhl Seminar Proceedings*, Vol. 05462, 2005, http://drops.dagstuhl.de/opus/volltexte/2006/524/pdf/05462.SWM.Paper.524.pdf.
11. World Wide Web Consortium (W3C), "Web services architecture," 2006, http://www.w3.org/TR/ws-arch/.
12. D. Garlan and B. Schmerl, "Architecture-driven modelling and analysis," in *Proceedings of the 11th Australian Workshop on Safety Related Programmable Systems, Conferences in Research and Practice in Information Technology*, Vol. 69, 2006, pp. 3-17.
13. C. Ouyang, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "From BPMN process models to BPEL web services," in *Proceedings of the 4th International Conference on Web Services*, 2006, pp. 285-293.
14. M. Voß, A. Hess, and B. Humm, "Towards a framework for large scale quality architecture," *Perspectives in Software Quality – 2nd International Conference on the Quality of Software Architectures*, 2006, pp. 52-58.
15. C. Pahl, "Layered ontology-based modelling of service-based software systems," *Information and Software Technology*, Vol. 49, 2007, pp. 838-850.
16. N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, Vol. 26, 2000, pp. 70-93.
17. U. Zdun, C. Hentrich, and S. Dustdar, "Modeling process-driven and service-oriented architectures using patterns and pattern primitives," *ACM Transactions on the*

Web, Vol. 1, 2007, pp. 14.1-14.44.
18. A. Albani, J. Dietz, and J. Zaha, "Identifying business components on the basis of an enterprise ontology," *Interoperability of Enterprise Software and Applications*, 2006, pp. 335-347.
19. O. Zimmermann, J. Koehler, and F. Leymann, "Architectural decision models as micro-methodology for service-oriented analysis and design," in *Proceedings of Workshop on Software Engineering Methods for Service Oriented Architecture*, Vol. 244, 2007, pp. 46-60.

Claus Pahl is a senior lecturer and the head of the Software and Systems Engineering group at Dublin City University. He is also the director the M.Sc. in Software Engineering degree programme. His research interests encompass software and service engineering, in particular the development and deployment of Web and Internet-based applications. He received his Ph.D. in Computer Science from the University of Dortmund, Germany. He is a member of the IEEE Computer Society and the German Association for Computer Science.

Wilhelm Hasselbring is a Professor of Software Engineering at the University of Kiel. His research interests include software engineering and distributed systems, particularly software architecture design and evaluation. He received his Ph.D. in Computer Science from the University of Dortmund, Germany. He's a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science.

Markus Voss is business unit head at Capgemini sd&m AG, Germany. From 2006-2007 he headed sd&m Research, the research and technology management division. His technical interests include enterprise architecture, systems integration and modern software engineering techniques. He received his Ph.D. in Computer Science from Karlsruhe University, Germany. He is member of the executive committee of the German Association for Computer Science.